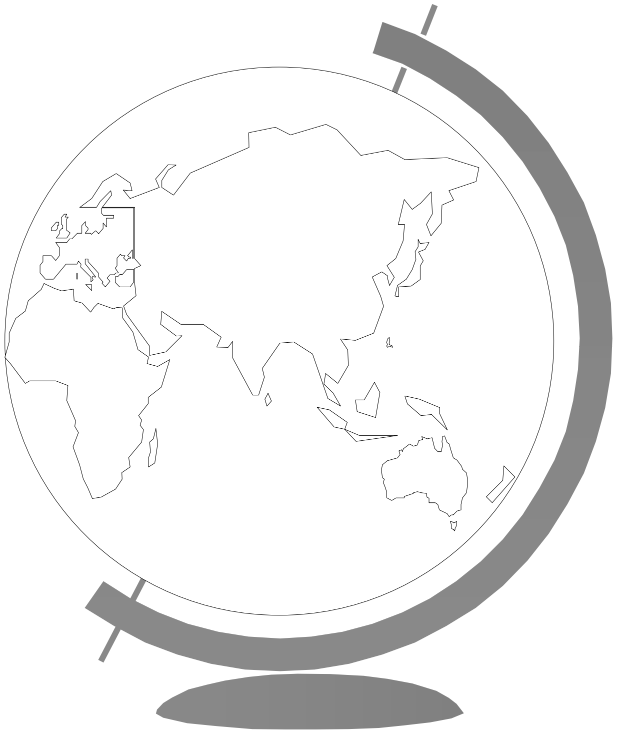


Intro to LAN/WAN

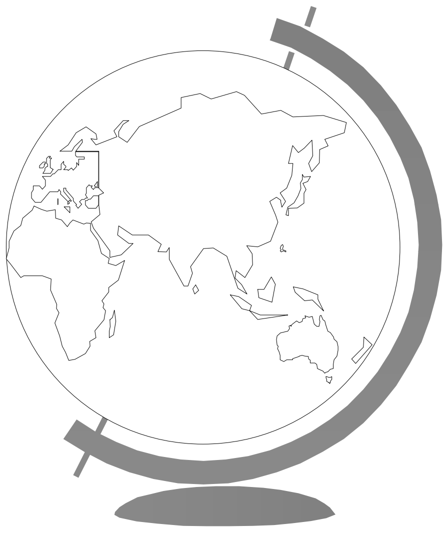
Transport Layer (Part III)

Transport Layer Topics

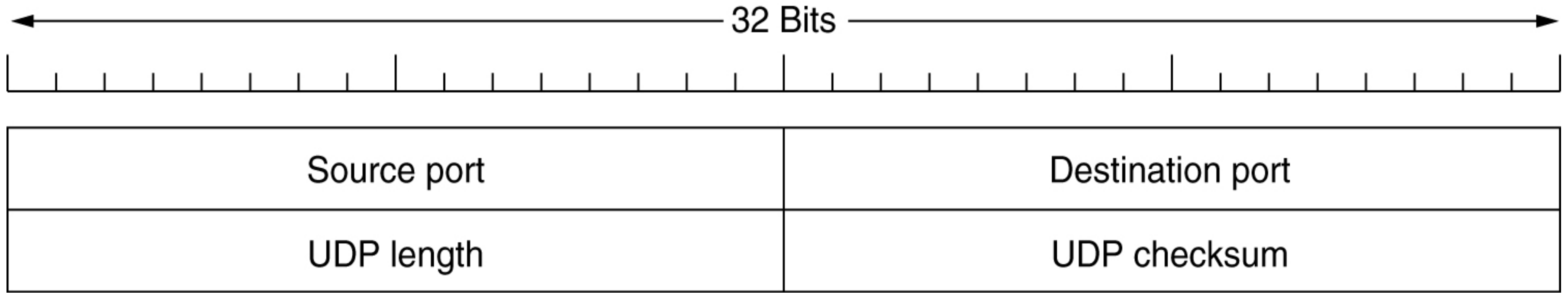
- ☞ Introduction (6.1)
- ☞ Elements of Transport Protocols (6.2)
- ☞ Internet Transport Protocols: TDP (6.5)
- ☞ Internet Transport Protocols: UDP (6.4) ←

UDP

- ☞ TCP:
  - Connection-oriented
  - Reliable, guarantees
  - ACKs
- ☞ UDP:
  - Connectionless
  - Unacknowledged, best effort
  - Basically IP with a short header added



UDP Segments



UDP header

- UDP segments:
  - 8-byte header followed by payload
- Two ports (source, destination.) identify endpoints
- At destination port: UDP pkt arrives, handed to process
- Process is associated with port in *BIND* socket call
- In fact: Key difference with raw IP is port associations with process

What UDP Do's and Don'ts

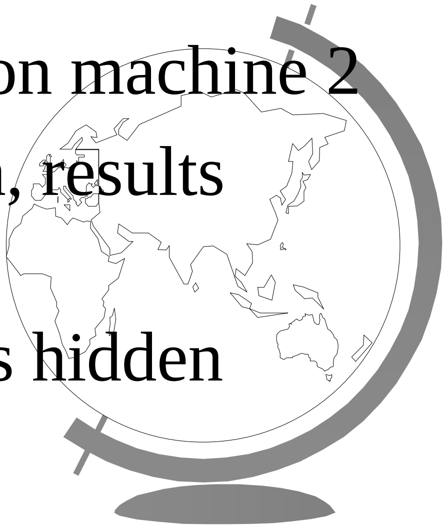
- ☞ UDP does no
  - – Flow control
  - – Error correction
  - – Retransmission, dest. process handles this
- ☞ UDP does:
  - – Multiplexing/demultiplexing via ports
- ☞ So, UDP has minimal features, applications do the rest
- ☞ UDP useful in client-server scenarios
  - – Short request, short response
  - – If request or response gets lost, client times out, sends again
  - – Example: DNS (chapter 7)

UDP Application: Remote Procedure Call (RPC)

- Sending messages to server and getting response is similar to function call in programming
- Both cases: start with one or more parameters, get result back
- Remote Procedure Call (Birrell and Nelson, 1984):
  - Attempt to cast interaction with a server as function call
  - Benefits: easier network programming
  - Example: function *get_IP_address(host_name)* which sends IP packet to a DNS server, gets IP address, hides networking

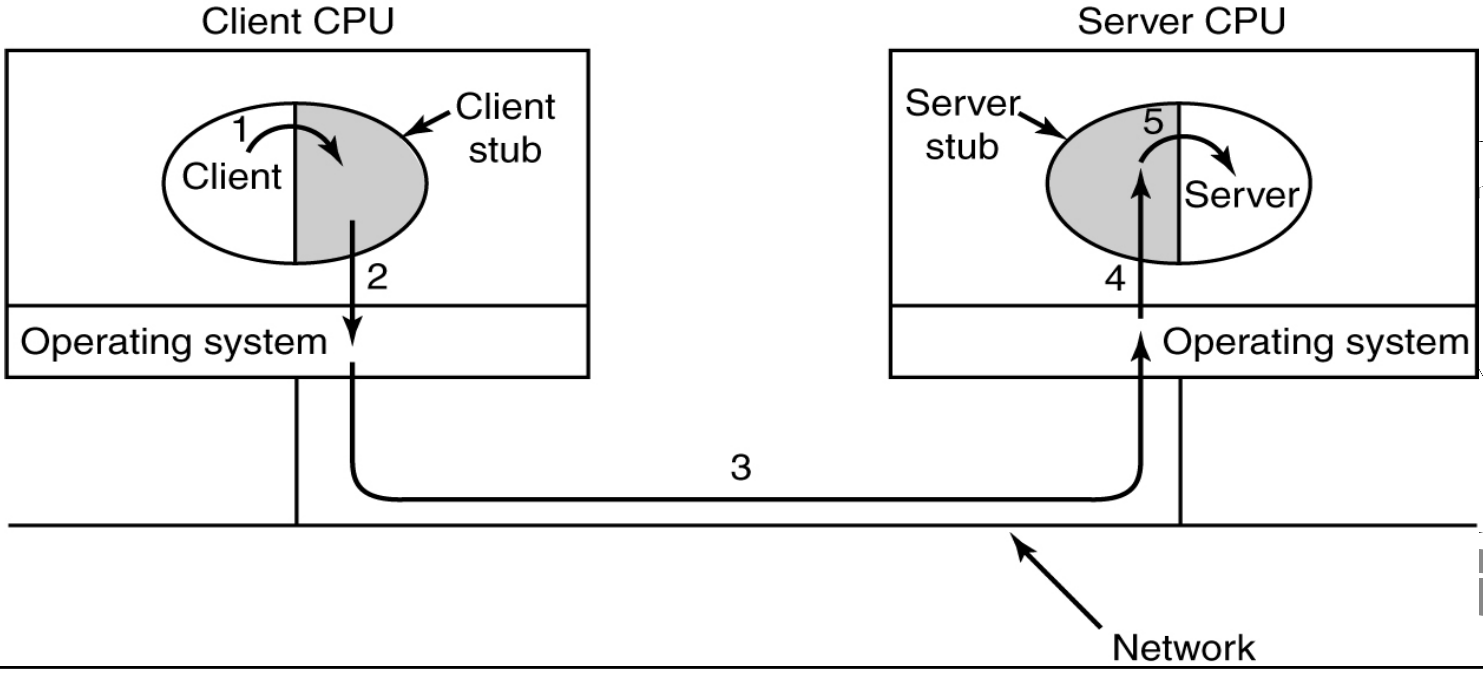
RPC Overview

- ☞ Overview
  - Program on local machines can call functions on a remote machine
  - Simply need to associate local calls to remote implementations
  - Networking is hidden
- ☞ Concrete example?
  - Machine 1 calls a procedure on Machine 2
  - Calling process on machine 1 hangs till execution on machine 2
  - Procedure and parameters sent in forward direction, results returned in backward direction
  - Programmer makes association once, networking is hidden



RPC Overview

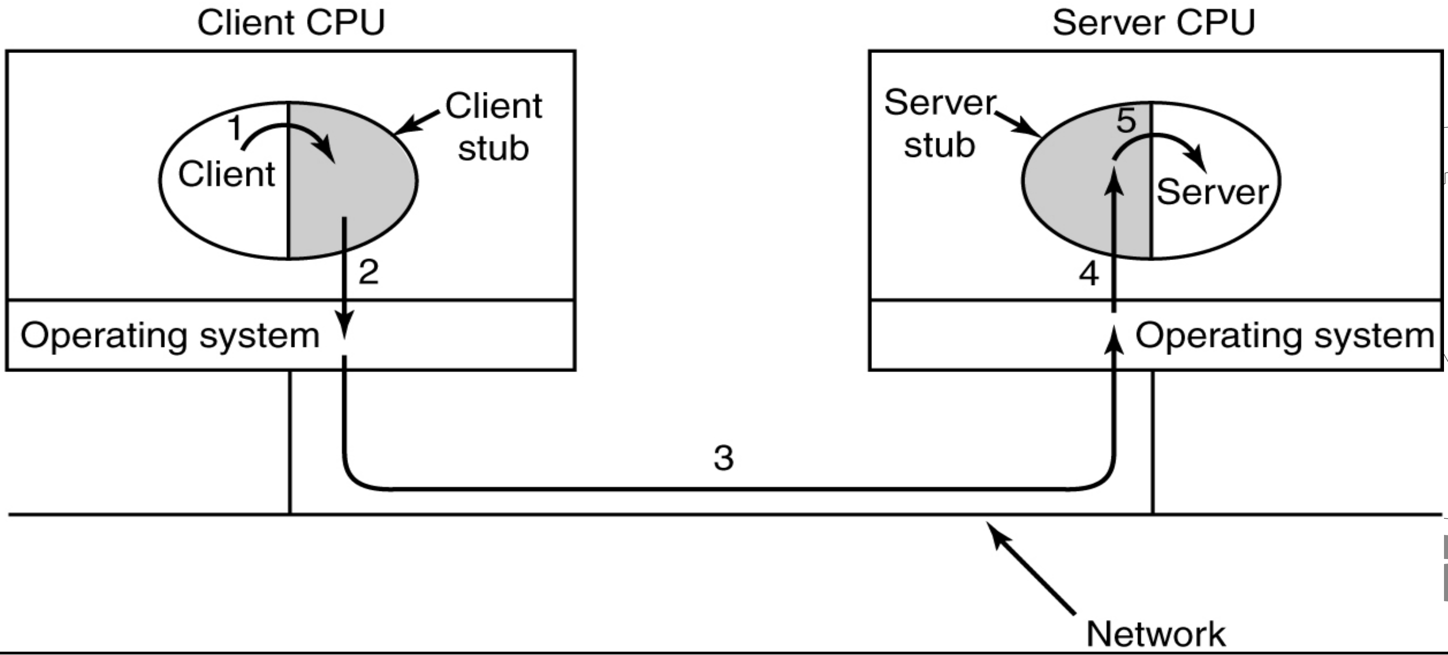
- ☞ Remote calls must resemble and feel like local
- ☞ Key idea:
  - – Client procedure bound with small library procedure on the clien called *client stub*, which represents server
  - – Server procedure bound with *server stub*



RPC Steps

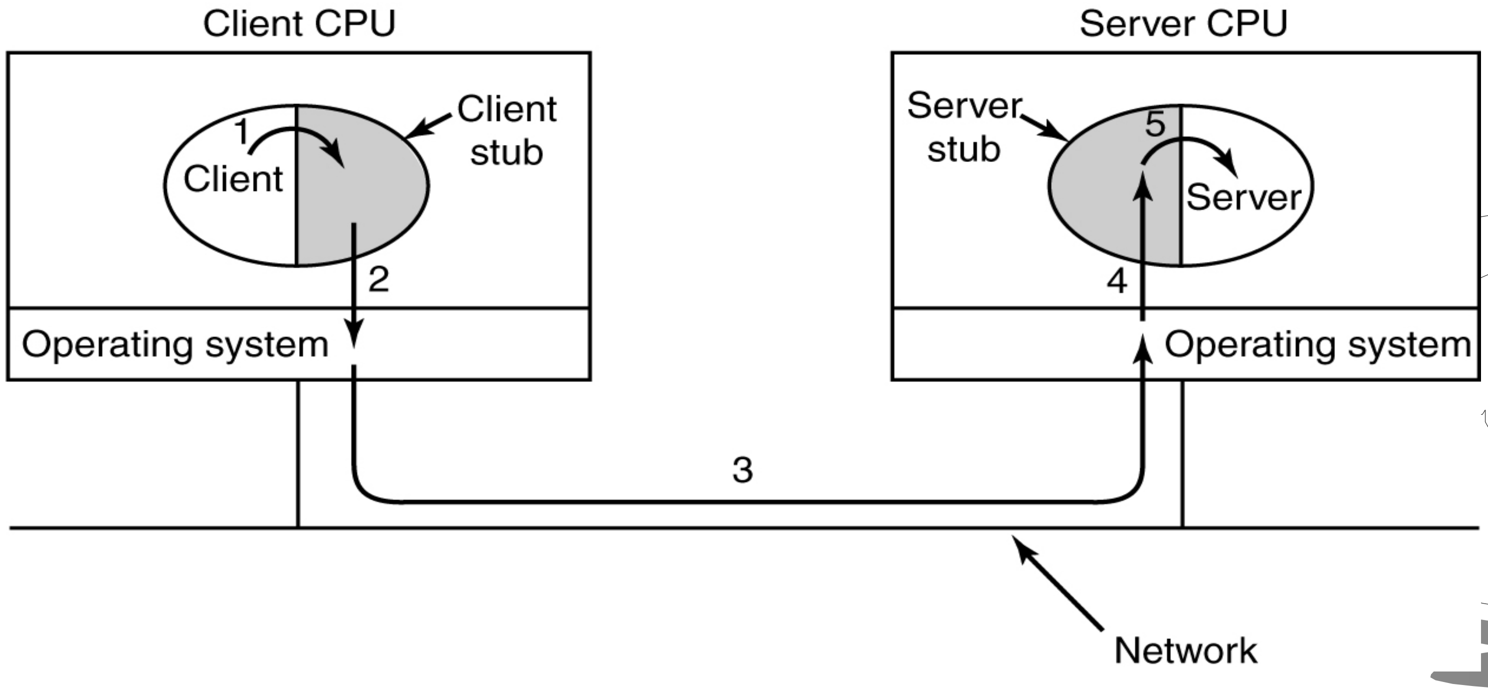
1. Client calls client stub
2. Marshalling: client packing parameters into a message and makes system call

☞ **Key note:** client simply makes local call with same name as remote server implementation



RPC Steps

3. Kernel sends message from client to server
4. Server kernel passes message to server stub
5. Server stub unmarshals parameters, calls server procedure



RPC Issues

☞ Pointer parameters:
- Pointers are basically reference to memory address
- Local use of pointers no problem, same address
- RPC: client and server different address spaces
- Can limit pointers to call-by-reference
- Call-by-reference fails if pointer to graph or complex data structure

☞ Other problems:
- Global variables, etc